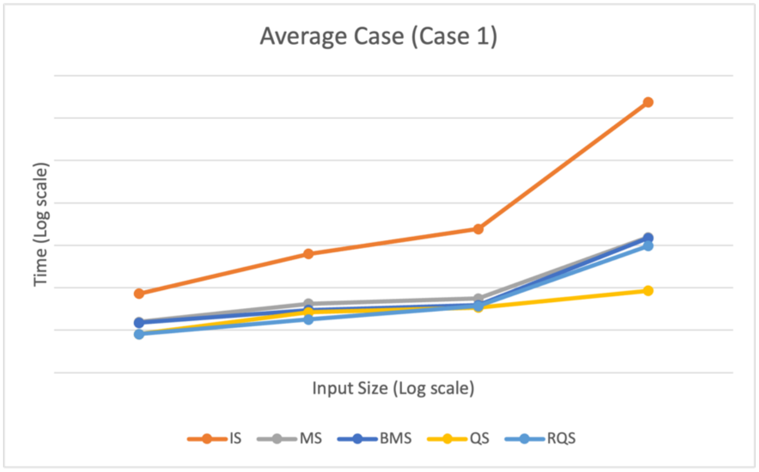
Algorithms PA 1 Report

B11901123 電機三 張甡源

1. Runtime and Memory usage of five sorting algorithms (Data from EDA union lab machine)

Input size	IS		MS		BMS		QS		RQS	
	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)
4000.case2	0.000124	5904	0.000523	5904	0.000848	5904	0.01507	6036	0.000622	5904
4000.case3	0.01416	5904	0.000868	5904	0.000853	5904	0.013043	5904	0.00066	5904
4000.case1	0.007274	5904	0.001555	5904	0.001486	5904	0.000808	5904	0.000808	5904
16000.case2	0.000109	6056	0.001621	6056	0.001361	6056	0.154598	6936	0.001338	6056
16000.case3	0.123757	6056	0.002352	6056	0.001516	6056	0.119648	6432	0.001871	6056
16000.case1	0.06307	6056	0.004185	6056	0.002979	6056	0.002632	6056	0.001785	6056
32000.case2	0.000112	6188	0.003188	6188	0.003196	6188	0.608229	8004	0.002547	6188
32000.case3	0.486918	6188	0.002747	6188	0.00275	6188	0.416485	6988	0.002181	6188
32000.case1	0.244382	6188	0.005604	6188	0.003892	6188	0.003405	6188	0.003686	6188
1000000.case2	0.000827	12144	0.071962	14004	0.067445	14000	591.706	72468	0.051071	12144
1000000.case3	474.147	12144	0.076993	14004	0.075289	14000	318.053	33012	0.052401	12144
1000000.case1	236.772	12144	0.154014	14004	0.147342	14000	0.085366	12144	0.097196	12144

2. Average Case (Case 1) Plotline



- Calculation of slope
  - Ex. Insertion sort average case

input size	IS
3.60206	-2.1382267
4.20412	-1.20017717
4.50515	-0.61193434
6	2.37433034

$$Average\ Slope = \frac{1}{3}\left[\frac{(-1.2)-(-2.14)}{4.2-3.6}+\frac{(-0.612)-(-1.2)}{4.5-4.2}+\frac{2.374-(-0.612)}{6-4.5}\right] \approx 1.84$$

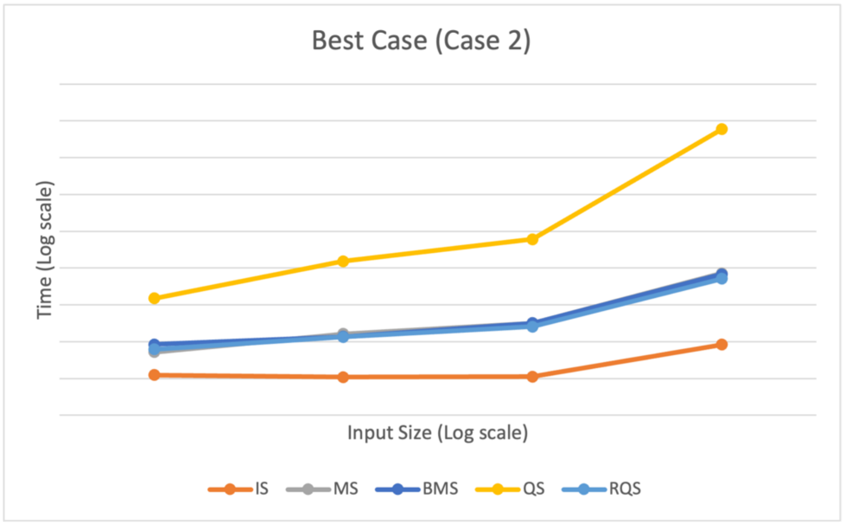
- Similarly, we can compute

Sorting method	Slope	Time complexity	Expected Slope
Insertion Sort	1.84	$O(n^2)$	2
Merge Sort	0.70	$O(nlgn)$	1~2
Bottom-up Merge Sort	0.65	$O(nlgn)$	1~2
Quick Sort	0.50	$O(nlgn)$	1~2
Randomized Quick Sort	0.86	$O(nlgn)$	1~2

- Calculation of expected slope, Ex. $O(n^2)$ is 2
  - For algorithm with $O(n^2)$, we have $T(n) \approx c \cdot n^2$, taking the logarithm of both sides, we have $\lg(T(n)) \approx 2(\lg(n)) + lgC$. Hence, we expect the slope to be 2.

- From the table above, we can conclude that all algorithms exhibit faster performance on average case inputs than their expected time complexities. Among these, insertion sort is the most time-consuming, as accurately reflected in its time complexity of $O(n^2)$. In conclusion, the trendline aligns closely with the one provided in the guidelines.

3. Best Case (Case 2) Plotline

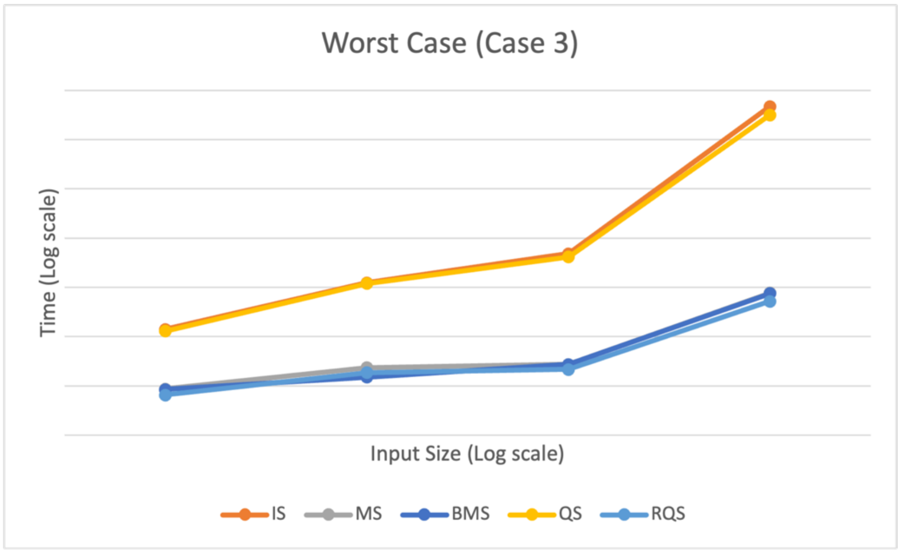


- Slope vs Expected Slope

Sorting method	Slope	Time complexity	Expected Slope
Insertion Sort	0.18	$O(n)$	1
Merge Sort	0.90	$O(nlgn)$	1~2
Bottom-up Merge Sort	0.82	$O(nlgn)$	1~2
Quick Sort	1.88	$O(nlgn)$	1~2
Randomized Quick Sort	0.78	$O(nlgn)$	1~2

- From the table above, we can conclude that all algorithms exhibit faster performance on best case inputs than their expected time complexities. Among these, insertion sort becomes the least time-consuming, as accurately reflected in its time complexity of $O(n)$. Another interesting point is quick sort significantly increases its slope, which will be discussed in the final section of this report. In conclusion, the trendline aligns closely with the one provided in the guidelines.

4. Worst Case (Case 3) Plotline



- Slope vs Expected Slope

Sorting method	Slope	Time complexity	Expected Slope
Insertion Sort	1.85	$O(n^2)$	2
Merge Sort	0.64	$O(nlgn)$	1~2
Bottom-up Merge Sort	0.75	$O(nlgn)$	1~2
Quick Sort	1.78	$O(n^2)$	2
Randomized Quick Sort	0.63	$O(nlgn)$	1~2

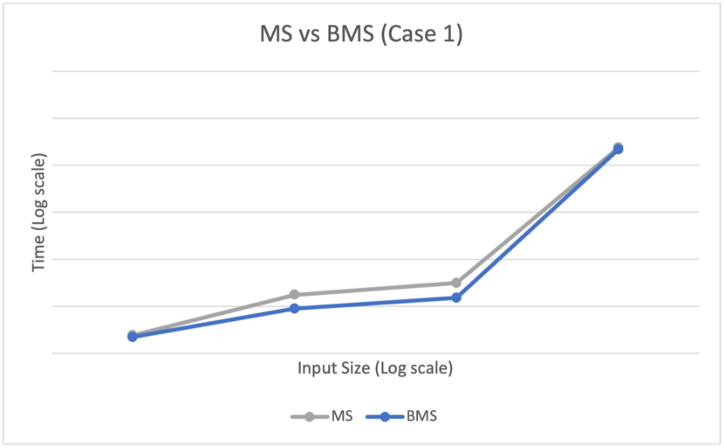
- From the table above, we can conclude that all algorithms demonstrate faster performance on worst case inputs compared to their expected time complexities. Notably, both insertion sort and quick sort have time complexities of $O(n^2)$, and their experimental slopes are significantly steeper than those of the other algorithms. An interesting observation is that quick sort performs almost identically slow in both the best and worst cases, while it is the fastest in random cases. This will be further discussed in the final section of the report. In conclusion, the trendline closely aligns with the one provided in the guidelines.

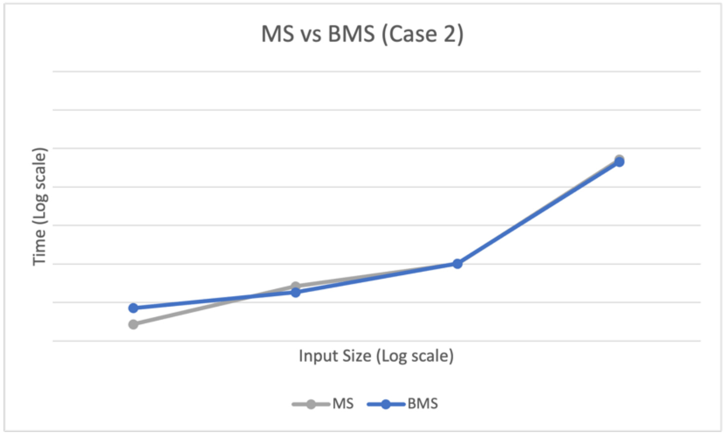
5. MS vs BMS

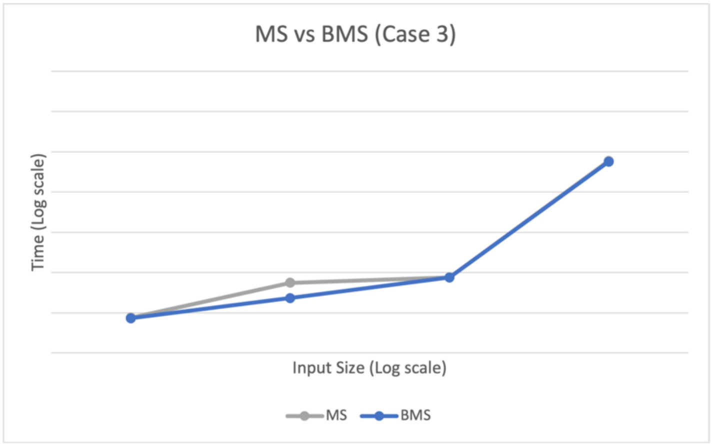
Algorithm	Best-case Time Complexity	Average-case Time Complexity	Worst-case Time Complexity
Merge Sort	$O(nlgn)$	$O(nlgn)$	$O(nlgn)$
Bottom-up Merge Sort	$O(nlgn)$	$O(nlgn)$	$O(nlgn)$

Although the time complexities of merge sort and bottom-up merge sort are both $O(n \log n)$, the experiment shows that BMS tends to be faster by 0.0001 to 0.001 seconds. This difference can be explained by the nature of each algorithm. MS uses recursion, which introduces overhead that can slow down the algorithm. In contrast, bottom-up merge sort (BMS) uses iteration, leading to slightly better performance.

For memory usage, both MS and BMS consume nearly the same amount of memory across all three cases. However, when the input size reaches 1,000,000, MS uses 4 KB more than BMS. This can be attributed to MS requiring additional memory for recursive stack space, increasing system pressure. In contrast, BMS eliminates stack by using loops, but the difference in memory consumption only becomes noticeable when the input size is significantly large, such as 1,000,000.





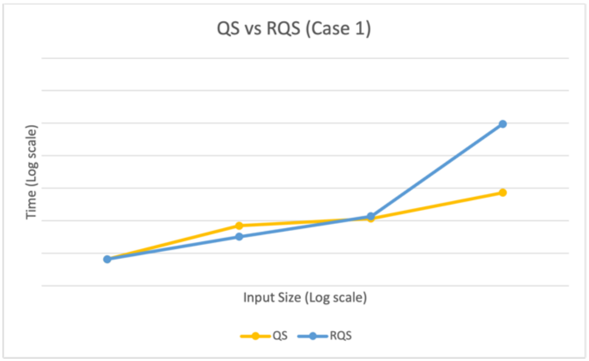


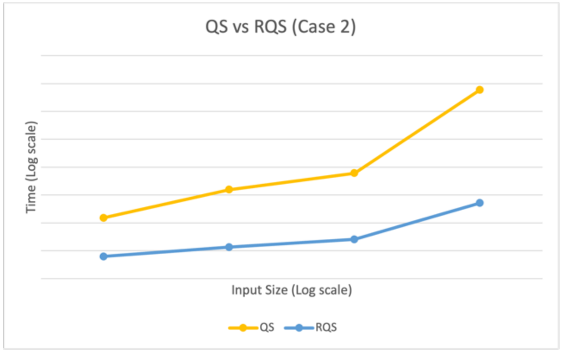
6. QS vs RQS

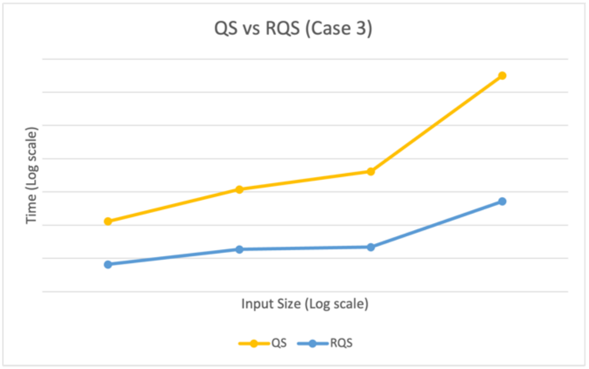
Algorithm	Best-case Time Complexity	Average-case Time Complexity	Worst-case Time Complexity
Quick Sort	$O(nlgn)$	$O(nlgn)$	$O(n^2)$
Randomized Quick Sort	$O(nlgn)$	$O(nlgn)$	$O(nlgn)$

The time complexities of quick sort and random quick sort are at best $O(nlgn)$, and at worst $O(n^2)$. The experiment shows that RQS tends to be faster by up to 500 seconds, especially when the input size reaches 1,000,000. This difference can be explained by the nature of each algorithm. RQS selects its pivot randomly, and this greatly reduces the chances of encoutering the worst-case scenario. On the other hand, in quick sort, we fix the pivot as the highest element, then in certain cases, it may create unbalanced partition, thus resulting in a slow running time. For the worst-case and best-case of 1,000,000 input for QS, the running time suddenly explodes to 591.706 seconds and 318.053 seconds from an average of 0.142 second for smaller inputs. This can interpreted that in that particular cases, since the partition is fixed from the largest indexed element, the partition is severely inbalanced, hence causing the running time to be soaring high.

For memory usage, both QS and RQS consume nearly the same amount of memory across cases with small inputs. However, when the input size reaches 1,000,000, QS consumes 72468 KB, 33012 KB, 12144 KB for three cases comparing to 12144 KB for all RQS case. This is due to the recursion depth caused by pivot selection in quick sort, the recursion depth can reach n for an input of size n. Whilst in RQS, due to the random selection of pivots that likely lead to more balanced partitions, the expected depth is n for an input of size lgn.







7. Data Structure and Other Findings
    1) Data Structure:
        a) Vector, a dynamic array where its size can change during runtime.
        b) IS, MS, BMS, QS, RQS data is passed by reference, ie. vector<int>& data
        c) Access and manipulating elements:
            i) Accessing elements: data[i] to randomly access using an index
            ii) Inserting elements: push_back(), insert(), or emplace() to add elements dynamically
            iii) Removing elements: erase(), pop_back()

    2) Other Findings:

When performing Quick Sort on 1,000,000 input elements, I observed that the best case scenario resulted in a significantly longer runtime than the worst case—591.706 seconds versus 318.053 seconds. Initially, this result seemed counterintuitive, but after further investigation, I realized that the issue lies in my choice of pivot. In my implementation, the highest-indexed element is always selected as the pivot. For already sorted input (best case), this leads to highly unbalanced partitions, resulting in the maximum possible number of recursive calls and significantly increasing the runtime.

Additionally, the unbalanced partitioning increases the recursion depth to nearly the size of the input (1,000,000), which causes the memory usage to spike to 591.706 KB in the best case. This deep recursion also triggered a segmentation fault (core dumped), which was resolved by increasing the stack size to 256 MB.

Overall, I learned about how to implement five sorting algorithms- IS, MS, BMS, QS, RQS, and compare their time and space performance.